Brief Announcement:
Self-stabilizing Systems in Spite of High Dynamics[*]

Karine Altisen[1], Stéphane Devismes[1], Anaïs Durand[2], Colette Johnen[3], and Franck Petit[4]

[1]VERIMAG, Univ. Grenoble Alpes, Grenoble, France
[2]LIMOS, Univ. Clermont Auvergne, Clermont-Ferrand, France
[3]LaBRI, Univ. de Bordeaux, Bordeaux, France
[4]LIP6, Sorbonne Univ., Paris, France

Abstract

We initiate research on self-stabilization in highly dynamic identified message-passing systems where dynamics is modeled using time-varying graphs (TVGs). More precisely, we address the self-stabilizing leader election problem in three wide classes of TVGs: the class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ of TVGs with temporal diameter bounded by Δ, the class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ of TVGs with temporal diameter quasi-bounded by Δ, and the class $\mathcal{TC}^{\mathcal{R}}$ of TVGs with recurrent connectivity only, where $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}}$. We first study conditions under which our problem can be solved. Precisely, we introduce the notion of size-ambiguity to show that the assumption on the knowledge of the number n of processes is central. Our results reveal that, despite the existence of unique process identifiers, any deterministic self-stabilizing leader election algorithm working in the TVG class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ or $\mathcal{TC}^{\mathcal{R}}$ cannot be size-ambiguous, justifying why our solutions for those classes assume the exact knowledge of n. We then present three self-stabilizing leader election algorithms for the TVG classes $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, and $\mathcal{TC}^{\mathcal{R}}$, respectively.

Keywords: Self-stabilization, time-varying graphs, leader election, speculation.

1 Introduction

Starting from an arbitrary configuration, a *self-stabilizing algorithm* [6] makes a distributed system reach within finite time a configuration from which its behavior is correct. Essentially, self-stabilizing algorithms tolerate *transient failures*, since by definition such failures last a finite time (as opposed to crash failures, for example) and their frequency is low (as opposed to intermittent failures). Even though self-stabilization is not inherently suited to handle *intermittent* and *permanent* failures, several works show that in many cases self-stabilization can still be achieved despite such faults occur. Indeed, strong forms of self-stabilization have been proposed to tolerate permanent failures, *e.g.*, *fault-tolerant self-stabilization* [2] to cope with process crashes, and *strict stabilization* [11] to withstand Byzantine failures. Furthermore, several self-stabilizing algorithms, *e.g.*, [5], withstand intermittent failures such as frequent lost, duplication, or reordering of messages. All these aforementioned works assume static communication networks. Nevertheless, self-stabilizing algorithms dedicated to arbitrary network topologies tolerate, up to a certain extent, some topological changes. Precisely, if topological changes are eventually detected locally at involved processes and if the frequency of such events is low enough, then they can be considered

[*]This study was partially supported by the French ANR projects ANR-16-CE40-0023 (DESCARTES) and ANR-16 CE25-0009-03 (ESTATE).

as transient faults. Actually, a number of works, *e.g.*, [4], use this kind of argument to claim that they are suited for the dynamic context. Furthermore, several approaches, like *superstabilization* [7] and *gradual stabilization* [1], aims at additionally providing specific countermeasures to efficiently treat topological changes when they are both spatially and timely sparse. However, all these aforementioned approaches, *e.g.*, [1, 4, 7], become totally ineffective when the frequency of topological changes drastically increase, in other words when topological changes are intermittent rather than transient. Actually, in the intermittent case, the network dynamics should be no more considered as an anomaly but rather as an integral part of the system nature.

Contribution. We study self-stabilizing leader election in *highly dynamic identified message-passing systems where the dynamics is modeled using Time-Varying Graphs* [3] (TVGs, for short) to obtain solutions tolerating both transient faults and high dynamics. We consider three wide classes of TVGs, respectively denoted by $\mathcal{TC}^{\mathcal{B}}(\Delta)$, $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, and $\mathcal{TC}^{\mathcal{R}}$, where $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}}$: $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is the class of TVGs with temporal diameter bounded by Δ [9], $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ is the class of TVGs with temporal diameter quasi-bounded by Δ (introduced here), and $\mathcal{TC}^{\mathcal{R}}$ is the class of TVGs with recurrent temporal connectivity [3].

We first study conditions under which our problem can be solved. Actually, our results show that the assumption on the knowledge of the number n of processes is central. To see this, we introduce the notion of *size-ambiguity*, which formalizes the fact that some subsets of processes do not share enough initial knowledge on n to detect that the system is not limited to themselves. In other words, such an ambiguity comes from the fact that n is only partially known by the processes. Our results show that, despite the existence of unique process identifiers, any deterministic self-stabilizing leader election algorithm working in the class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ or $\mathcal{TC}^{\mathcal{R}}$ cannot be size-ambiguous. Hence, to make the problem solvable in those classes, we assume each process knows exactly n.

We then propose self-stabilizing leader election algorithms for the three considered classes. In more detail, we present a self-stabilizing leader election algorithm for Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ with a stabilization time of at most 3Δ rounds, assuming every process knows Δ, yet using no information on n. This in particular shows that our necessary condition is tight. Then, we propose a self-stabilizing leader election algorithm for Class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ assuming every process knows Δ and n. In general, stabilization time cannot be bounded in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$; nevertheless its stabilization time in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is at most 2Δ rounds. In other words, this algorithm is *speculative* (in the sense of [8]) since it exhibits better performances in a subset of more probable executions. Finally, we propose a self-stabilizing leader election algorithm for Class $\mathcal{TC}^{\mathcal{R}}$, where only n is known, yet requiring unbounded local memories. Again, in general, stabilization time cannot be bounded in $\mathcal{TC}^{\mathcal{R}}$, yet the algorithm is speculative since its stabilization time in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is at most $\Delta + 1$ rounds.

Roadmap. Due to the lack of space, we only present here our three algorithms. All our results are available in an online technical report.[1] In Section 2, we briefly present the model. In Section 3, we outline our algorithms. We give perspectives in Section 4.

2 Preliminaries

Time-varying Graphs. A *time-varying graph* [3] (TVG for short) is a tuple $\mathcal{G} = (V, E, \mathcal{T}, \rho)$ where V is a set of nodes, E is a set of arcs between pairwise nodes, $\mathcal{T}$ is an interval over $\mathbb{N}^*$, and $\rho : E \times \mathcal{T} \to \{0, 1\}$ is the *presence* function that indicates whether or not a given arc exists at a given time. We denote by $o_{\mathcal{T}} = \min \mathcal{T}$ the first instant in $\mathcal{T}$. The *snapshot* of $\mathcal{G}$ at time $t \in \mathcal{T}$ is the graph $G_t = (V, \{e \in E : \rho(e, t) = 1\})$. A *journey* is a sequence of ordered pairs $\mathcal{J} = (e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k)$ where

[1] https://hal.archives-ouvertes.fr/hal-02376832/

$\forall i \in \{1, \ldots, k\}$, $e_i = (p_i, q_i) \in E$ satisfies $\rho(e_i, t_i) = 1$ and $i < k \Rightarrow q_i = p_{i+1} \wedge t_i < t_{i+1}$. The *temporal length* of a journey $\mathcal{J} = (e_1, t_1), (e_2, t_2), \ldots, (e_k, t_k)$ is equal to $t_k - t_1 + 1$. The *temporal distance* from p to q at time t is the first arrival time of a journey from p to q whose starting time is at the earliest time t. The *temporal diameter* of $\mathcal{G}$, is the maximum temporal distance between any two nodes of $\mathcal{G}$ at any time.

TVG Classes. We consider the following TVG classes.
Class $\mathcal{TC}^{\mathcal{R}}$ (Recurrent Temporal Connectivity): at any point in time, every node can reach all the others through a journey.
Class $\mathcal{TC}^{\mathcal{Q}}(\Delta)$ with $\Delta \in \mathbb{N}^*$ (Quasi Bounded Temporal Diameter): at any point in time, every node can eventually reach each other node through a journey of temporal length at most Δ.
Class $\mathcal{TC}^{\mathcal{B}}(\Delta)$ with $\Delta \in \mathbb{N}^*$ (Bounded Temporal Diameter): at any point in time, the temporal distance between any pair of two nodes is at most Δ, *i.e.*, the temporal diameter is bounded by Δ.

Notice that, $\forall \Delta \in \mathbb{N}^*$, $\mathcal{TC}^{\mathcal{B}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{Q}}(\Delta) \subseteq \mathcal{TC}^{\mathcal{R}}$.

Computational Model. We assume a distributed system made of a set of n processes, denoted by V. Each process p has a unique identifier denoted by $id(p)$ and taken in an arbitrary domain $IDSET$ totally ordered by $<$. We denote by ℓ the process of minimum identifier. A *distributed algorithm* $\mathcal{A}$ is a collection of n local algorithms $\mathcal{A}(p)$, one per process $p \in V$. The *state* in $\mathcal{A}$ of each process $p \in V$ is defined by the values of its variables in $\mathcal{A}(p)$. Some variables may be constants in which case their values are predefined. A *configuration* of $\mathcal{A}$ for V is a vector of n components $(s_1, s_2, \ldots, s_n)$, where s_1 to s_n represent the states of the processes in V.

Processes execute their local algorithms in *synchronous rounds*. For every $i > 0$, the communication network at Round i is defined by $G_{o_{\mathcal{T}}+i-1}$, *i.e.*, the snapshot of $\mathcal{G}$ after $i - 1$ instants elapse from the initial time $o_{\mathcal{T}}$. For any (synchronous) round $i \geq 1$, the system moves from the current configuration γ_{i-1} to some configuration γ_i. Such a move is atomically performed by every process $p \in V$ according to the following three steps, defined in its local algorithm $\mathcal{A}(p)$: (1) p sends a message consisting of all or a part of its local state in γ_{i-1}, (2) p receives all messages sent by its neighbors in $G_{o_{\mathcal{T}}+i-1}$, and (3) p computes its state in γ_i.

Self-stabilizing Leader Election. In identified networks, leader election usually consists in making the processes agree on one of the identifiers held by processes. The identifier of the elected process is stored at each process p in an output variable, denoted here by $lid(p)$. In the self-stabilizing context, variables lid may be initially corrupted; in particular some of them may be initially assigned to *fake IDs*, a value $v \in IDSET$ such that v is not assigned as a process identifier in the system. Despite such fake IDs, the goal of a self-stabilizing algorithm is to make the system converge to a configuration from which a unique process is forever adopted as leader by all processes. Our solutions always elect the process of lowest ID, ℓ. To that goal, our algorithms implement two main mechanisms: one for removing all fake IDs from the system, and the other for computing the minimum value among the remaining IDs.

3 Algorithms

Algorithm for $\mathcal{TC}^{\mathcal{B}}(\Delta)$. Recall that we assume every process knows Δ. Then, each process p maintains two variables: the output $lid(p)$ eventually contains the ID of the leader; $ttl(p)$ represents the *degree of mistrust* of p in $lid(p)$ and allows to eliminate messages containing fake IDs. The value $ttl(p)$ increases at each round if p does not receive a message; otherwise it is updated thanks to the received messages. $ttl(p)$ can increase up to $2\Delta - 1$. Process p never increases $ttl(p)$ from $2\Delta - 1$ to 2Δ; instead it locally *resets* and declares itself as the leader: $lid(p) := id(p)$ and $ttl(p) := 0$.

At each round i, p first sends its leader ID together with its degree of mistrust. Then, p selects the received message $\langle id, ttl\rangle$ which is minimum using the lexicographic order (*i.e.*, the message with the lowest ID and with the lowest *ttl* to break ties, if any). If *id* is smaller than $lid(p)$, p updates its leader $lid(p)$. If $id = lid(p)$, it updates the $ttl(p)$ by taking the smallest value between $ttl(p)$ and *ttl* (in this way, p may decrease its mistrust in $lid(p)$). In either case, $ttl(p)$ is then incremented if $lid(p) \neq id(p)$. Finally, if $lid(p) \geq id(p)$, p systematically resets. So, if p believes to be the leader at the end of Round i (*i.e.*, $lid(p) = id(p)$), then it sends its own ID together with a degree of mistrust 0 at the beginning of the next round, $i + 1$.

The resets allow to remove all fake IDs within at most 2Δ rounds. From that time, $(lid(\ell), ttl(\ell)) = (id(\ell), 0)$ forever. So, after 2Δ rounds, ℓ sends $\langle id(\ell), 0\rangle$ at each round and all processes will receive messages $\langle id(\ell), d\rangle$, with $d \leq \Delta < 2\Delta$ (since $\Delta \in \mathbb{N}^*$), at least every Δ rounds since the temporal diameter is upper bounded by Δ. Thus, within at most Δ additional rounds, they will all adopt ℓ as leader and never more reset, ensuring that ℓ will remain the leader forever. Hence, this algorithm is a self-stabilizing leader election algorithm for $\mathcal{TC}^{\mathcal{B}}(\Delta)$ and its stabilization time is at most 3Δ rounds.

Algorithm for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. We assume now that every process knows both Δ and n. Then, each process p uses a variable $members(p)$ to collect IDs. Actually, $members(p)$ is a FIFO queue containing at most n pairs $\langle id, t\rangle$, where *id* is an identifier and t is a timestamp, *i.e.*, an integer value less than or equal to Δ. At each round i, p sends all pairs $\langle id, t\rangle$ of $members(p)$ such that $t < \Delta$ at the end of Round $i - 1$. (The timestamps allow to eventually remove all fake IDs.) Then, p updates $members(p)$ by calling function $insert$ on each received pair $\langle id, t\rangle$ such that $id \neq id(p)$.

The insertion function $insert$ works as follows: if *id* already appears in $members(p)$, then the old pair tagged with *id* is removed first from the queue and in either case, $\langle id, t\rangle$ is appended at the tail of the queue. In particular, since the size of $members(p)$ is limited, if the queue is full, its head is removed to make room for the new value. Using this FIFO mechanism, initial spurious values eventually vanish from $members(p)$.

After all received pairs have been managed, the timestamps of all pairs in the queue are incremented and then, $\langle id(p), 0\rangle$ is systematically inserted at the tail of the queue. This mechanism ensures two main properties. First, every timestamp associated to a fake ID in a variable $members$ is eventually forever greater than or equal to Δ; and consequently, eventually no message containing fake IDs is sent. Second, by definition of $\mathcal{TC}^{\mathcal{Q}}(\Delta)$, for every two distinct processes p and q, there are journeys of length at most Δ infinitely often, so each process p regularly receives messages containing $id(q)$ with timestamps smaller than Δ. Thus, eventually $members(p)$ exactly contains all IDs of the networks. Now, at the end of each round, p updates its leader variable with the smallest ID in $members(p)$. Hence, the process of lowest ID, ℓ, is eventually elected. Thus, this algorithm is a self-stabilizing leader election for $\mathcal{TC}^{\mathcal{Q}}(\Delta)$.

In general, stabilization time cannot be bounded in $\mathcal{TC}^{\mathcal{Q}}(\Delta)$. Yet, if our algorithm is deployed on a TVG of class $\mathcal{TC}^{\mathcal{B}}(\Delta)$, then after Δ rounds, processes no more receives messages with fake IDs. For that point, the identity of every process will be inserted into queue of all processes within at most Δ rounds, which ensures stabilization time of at most 2Δ rounds. Therefore, this algorithm is speculative.

Algorithm for $\mathcal{TC}^{\mathcal{R}}$. Similarly to the previous algorithm, each process p uses a variable $members(p)$ to collect IDs. However, this time, only n is known and $members(p)$ is a map that can contain up to n IDs, each of them being associated with a timestamp.

At each round i, p sends the content of $members(p)$. Then, p updates $members(p)$ by calling function $insert$ on each received pair $\langle id, t\rangle$ such that $id \neq id(p)$. The function $insert$ works as follows: if *id* already appears in $members(p)$, then the associated timestamp is updated by keeping the smallest value. Otherwise, p tries to insert $\langle id, t\rangle$ in the map. Actually, $\langle id, t\rangle$ is inserted in the map if the map is not full or t is smaller than the greatest timestamp in the map. In this latter case, $\langle id, t\rangle$ overwrites any value having this timestamp in $members(p)$. This overwriting mechanism allows to eventually remove all fake IDs from $members(p)$, since their timestamps will regularly increase. After $members(p)$ has been updated, all timestamps of $members(p)$ are incremented and then, $\langle id(0), 0\rangle$ is systematically inserted in the map.

Actually, our algorithm guarantees two main properties. First, at the beginning of any round i, any timestamp associated to a fake ID is greater than or equal to $i - 1$. Second, by definition of $\mathcal{TC}^{\mathcal{R}}$, at any point in time, every process can reach all the others through a journey. The key property is then to show that if some broadcast initiated by process p reaches a process q at Round i, then the value of the timestamp in the message is small enough to ensure the insertion of $id(p)$ into $members(q)$. These two properties ensure that eventually $members(p)$ exactly contains all IDs of the network. Now, at the end of each round, p updates its leader variable with the smallest ID in $members(p)$. Hence, ℓ is eventually elected. Thus, this algorithm is a self-stabilizing leader election for $\mathcal{TC}^{\mathcal{R}}$.

In general, stabilization time cannot be bounded in $\mathcal{TC}^{\mathcal{R}}$. Yet, assume that the algorithm is deployed on a TVG of class $\mathcal{TC}^{\mathcal{B}}(\Delta)$. After the first round, every process p sends the pair $\langle id(p), 0\rangle$ at each round. Thus, any process q receives a pair $\langle id(p), t\rangle$ with $t < \Delta$ not later than Round $\Delta + 1$. Now, when this event occurs, each fake ID in $members(q)$ has a timestamp greater than t. So, $id(p)$ is definitely inserted into $members(q)$. Hence, the algorithm is speculative since its stabilization time in $\mathcal{TC}^{\mathcal{B}}(\Delta)$ is at most $\Delta + 1$ rounds.

4 Perspectives

Beyond extending our results to other particular problems, our future work will focus on studying expressiveness of self-stabilization in TVGs. To that goal, we plan to first investigate broadcast problems, again in very general TVG classes. Indeed, coupled with our leader election solutions, they should allow to build generic transformers, following, for example, the approach proposed in [10].

References

[1] Karine Altisen, Stéphane Devismes, Anaïs Durand, and Franck Petit. Gradual stabilization. *JPDC*, 123:26–45, 2019.

[2] Joffroy Beauquier and Synnöve Kekkonen-Moneta. On FTSS-solvable distributed problems. In *PODC*, page 290, 1997.

[3] Arnaud Casteigts, Paola Flocchini, Walter Quattrociocchi, and Nicola Santoro. Time-varying graphs and dynamic networks. *IJPEDS*, 27(5):387–408, 2012.

[4] Ajoy K. Datta and Lawrence L. Larmore. Self-stabilizing leader election in dynamic networks. *Theory Comput. Syst.*, 62(5):977–1047, 2018.

[5] Sylvie Delaët, Bertrand Ducourthial, and Sébastien Tixeuil. Self-stabilization with r-operators revisited. *JACIC*, 3(10):498–514, 2006.

[6] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[7] Shlomi Dolev and Ted Herman. Superstabilizing protocols for dynamic distributed systems. *Chicago Journal of Theoretical Computer Science*, 1995.

[8] S. Dubois and R. Guerraoui. Introducing speculation in self-stabilization: an application to mutual exclusion. In *PODC*, pages 290–298, 2013.

[9] Carlos Gómez-Calzado, Arnaud Casteigts, Alberto Lafuente, and Mikel Larrea. A connectivity model for agreement in dynamic systems. In *Euro-Par*, pages 333–345, 2015.

[10] Shmuel Katz and Kenneth J. Perry. Self-stabilizing extensions for message-passing systems. *Distributed Computing*, 7(1):17–26, 1993.

[11] Mikhail Nesterenko and Anish Arora. Dining philosophers that tolerate malicious crashes. In *ICDCS*, pages 191–198, 2002.